LAB2 CODE

Team 1

TEAM MEMBERS:

ESWAR VALLURU-27

BHAVYA SRI SIRASANAGANDLA-21

MANIDEEP NADIMPELLI -16

Question 1:

%tensorflow\_version 1.15  
import tensorflow as tf  
import keras  
from keras.datses import boston\_housing  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_plab2\_selection import train\_test\_split  
from keras import Sequential  
from keras.layers import Dense  
from keras.callbacks import TensorBoard  
from time import time  
from keras.optimizers import adam  
  
(x\_tr,y\_tr),(x\_te,Y\_te)=boston\_housing.load\_data()  
  
lrate=0.01  
s=adam(lr=lrate)  
  
model\_plab2=Sequential()  
model\_plab2.add(Dense(20,activation='relu',input\_shape=[x\_tr.shape[1]]))  
model\_plab2.add(Dense(1,activation='softmax'))  
  
model\_plab2.compile(optimizer='rmsprop',loss='mae',metrics=['mae'])  
  
tensorboard=TensorBoard(log\_dir="logs/{}".format(time()))  
model\_plab2.fit(x\_tr,y\_tr, batch\_size=40,epochs=50,callbacks=[tensorboard],validation\_data=(x\_te,Y\_te))  
  
model\_plab2.evaluate(x\_te,Y\_te)

Question 2:

import keras  
from keras.model\_plab2s import Sequential  
from keras.layers import Dense  
import numpy asnupy  
import pandas as paid  
from sklearn.model\_plab2\_selection import train\_test\_split  
from keras.callbacks import TensorBoard  
from time import time  
  
"""READ THE DATA USING read\_csv()"""  
  
datse = paid.read\_csv('heart.csv',index\_col=0)  
datse.astype(float)  
# Normalize values to range [0:1]  
datse /= datse.max()  
  
y = datse['target']  
X = datse.drop(['target'], axis = 1)  
X\_tr, X\_te, Y\_tr, Y\_te = train\_test\_split(X, y, test\_size = 0.33, random\_state = 0)  
  
np.random.seed(155)  
model\_plab2 = Sequential() # create model\_plab2  
model\_plab2.add(Dense(40, input\_dim=12, activation='relu')) # hidden layer  
model\_plab2.add(Dense(20, input\_dim=40, activation='relu'))  
model\_plab2.add(Dense(1, activation='sigmoid')) # output layer  
#model\_plab2.add(Dense(1, activation='softmax')) # output layer  
  
model\_plab2.compile(loss= keras.losses.binary\_crossentropy,optimizer=keras.optimizers.adamax(),metrics=['accuracy'])  
  
tensorborad = TensorBoard(log\_dir="logs/{}".format(time()))  
  
history = model\_plab2.fit(X\_tr, Y\_tr,batch\_size=30,epochs=50,verbose=1,  
 validation\_data=(X\_te, Y\_te), callbacks=[tensorborad])  
# Change batch size  
# history = model\_plab2.fit(X\_tr, Y\_tr,batch\_size=200,epochs=50,verbose=1,  
 # validation\_data=(X\_te, Y\_te), callbacks=[tensorborad])  
# Change epochs   
# history = model\_plab2.fit(X\_tr, Y\_tr,batch\_size=200,epochs=30,verbose=1,  
 # validation\_data=(X\_te, Y\_te), callbacks=[tensorborad])  
ypre = model\_plab2.predict\_classes(X\_te)  
  
score = model\_plab2.evaluate(X\_te, Y\_te, verbose=0)  
print('Loss:', score[0])  
print('Accuracy:', score[1])

Question 3:

import math   
import matplotlib.pyplot as pet   
import scipy   
import cv2   
import numpy asnupy   
import glob   
import os   
import pandas as paid   
import tensorflow as tf   
import itertools  
import random  
from random import shuffle   
from tqdm import tqdm   
from PIL import Image  
from scipy import ndimage  
from pathlib import Path  
from sklearn.metrics import classification\_report, confusion\_matrix  
from sklearn import metrics  
np.random.seed(1)  
  
from google.colab import drive  
drive.mount('/content/drive')  
  
from keras.preprocessing.image import ImageDataGenerator  
from keras.model\_plab2s import Sequential  
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense  
from keras.callbacks import Model\_plab2Checkpoint, EarlyStopping  
  
train\_dir = Path('/content/drive/My Drive/Colab Notebooks/10-monkey-species/training/training/')  
test\_dir = Path('/content/drive/My Drive/Colab Notebooks/10-monkey-species/validation/validation/')  
  
#lable info  
cols = ['Lable','Latin Name', 'Common Name','Train Images', 'Validation Images']  
lab = paid.read\_csv("/content/drive/My Drive/Colab Notebooks/10-monkey-species/monkey\_lab.txt", names=cols, skiprows=1)  
lab  
  
lab = lab['Common Name']  
lab  
  
def image\_show(num\_image,lable):  
 for i in range(num\_image):  
 imgdir = Path('/content/drive/My Drive/Colab Notebooks/10-monkey-species/training/training/' + lable)  
 #print(imgdir)  
 imgfile = random.choice(os.listdir(imgdir))  
 #print(imgfile)  
 img = cv2.imread('/content/drive/My Drive/Colab Notebooks/10-monkey-species/training/training/'+ lable +'/'+ imgfile)  
 # print(img.shape)  
 #print(lable)  
 pet.figure(i)  
 pet.imshow(img)  
 pet.title(imgfile)  
 pet.show()  
  
print(lab[4])  
image\_show(3,'n4')  
  
LR = 1e-3  
height=150  
width=150  
channels=3  
seed=1337  
batch\_size = 64  
num\_classes = 10  
epochs = 100  
data\_augmentation = True  
num\_predictions = 20  
  
# Training generator  
train\_datagen = ImageDataGenerator(  
 rescale=1./255,  
 rotation\_range=40,  
 width\_shift\_range=0.2,  
 height\_shift\_range=0.2,  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True,  
 fill\_mode='nearest')  
  
train\_generator = train\_datagen.flow\_from\_directory(train\_dir,   
 target\_size=(height,width),  
 batch\_size=batch\_size,  
 seed=seed,  
 shuffle=True,  
 class\_mode='categorical')  
  
# Test generator  
test\_datagen = ImageDataGenerator(rescale=1./255)  
validation\_generator = test\_datagen.flow\_from\_directory(test\_dir,   
 target\_size=(height,width),   
 batch\_size=batch\_size,  
 seed=seed,  
 shuffle=False,  
 class\_mode='categorical')  
  
train\_num = train\_generator.samples  
validation\_num = validation\_generator.samples  
  
model\_plab2 = Sequential()  
model\_plab2.add(Conv2D(32, (3, 3), input\_shape=(150, 150, 3)))  
model\_plab2.add(Activation('relu'))  
model\_plab2.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model\_plab2.add(Conv2D(32, (3, 3)))  
model\_plab2.add(Activation('relu'))  
model\_plab2.add(MaxPooling2D(pool\_size=(2, 2)))  
  
model\_plab2.add(Conv2D(64, (3, 3), padding='same'))  
model\_plab2.add(Activation('relu'))  
model\_plab2.add(Conv2D(64, (3, 3)))  
model\_plab2.add(Activation('relu'))  
model\_plab2.add(MaxPooling2D(pool\_size=(2, 2)))  
model\_plab2.add(Dropout(0.25))  
  
model\_plab2.add(Flatten())  
model\_plab2.add(Dense(512))  
model\_plab2.add(Activation('relu'))  
model\_plab2.add(Dropout(0.5))  
model\_plab2.add(Dense(num\_classes))  
model\_plab2.add(Activation('softmax'))  
  
  
  
model\_plab2.compile(optimizer='adam',  
 loss='categorical\_crossentropy',  
 metrics=['acc'])  
model\_plab2.summary()  
  
filepath=str(os.getcwd()+"/model\_plab2.h5f")  
checkpoint = Model\_plab2Checkpoint(filepath, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')  
# = EarlyStopping(monitor='val\_acc', patience=15)  
callbacks\_list = [checkpoint]#, stopper]  
  
history = model\_plab2.fit\_generator(train\_generator,  
 steps\_per\_epoch= train\_num // batch\_size,  
 epochs=epochs,  
 validation\_data=train\_generator,  
 validation\_steps= validation\_num // batch\_size,  
 callbacks=callbacks\_list,   
 verbose = 1  
 )  
  
acc = history.history['acc']  
val\_acc = history.history['val\_acc']  
loss = history.history['loss']  
val\_loss = history.history['val\_loss']  
epochs = range(1, len(acc) + 1)  
  
pet.title('Training and validation accuracy')  
pet.plot(epochs, acc, 'red', lable='Training acc')  
pet.plot(epochs, val\_acc, 'blue', lable='Validation acc')  
pet.legend()  
  
pet.figure()  
pet.title('Training and validation loss')  
pet.plot(epochs, loss, 'red', lable='Training loss')  
pet.plot(epochs, val\_loss, 'blue', lable='Validation loss')  
  
pet.legend()  
  
pet.show()  
  
print(metrics.classification\_report(validation\_generator.classes, Ypre\_classes,target\_names=lab))

Question 4:

import re  
import matplotlib.pyplot as plot  
import pandas as paid  
from keras import Sequential  
from keras.constraints import maxnorm  
from keras.layers import Embedding, Conv1D, Dropout, MaxPooling1D, Flatten, Dense, LSTM, GlobalMaxPooling1D  
from keras.optimizers import SGD  
from keras.utils import to\_categorical  
from keras\_preprocessing.sequence import pad\_sequences  
from keras\_preprocessing.text import Tokenizer  
from sklearn.model\_plab2\_selection import train\_test\_split  
from sklearn.preprocessing import LableEncoder  
  
tr\_mv\_df = paid.read\_csv('train.tsv', delimiter='\t', encoding='utf-8')  
te\_mv\_df = paid.read\_csv('test.tsv', delimiter='\t', encoding='utf-8')  
  
tr\_mv\_df = tr\_mv\_df.drop(columns=['PhraseId', 'SentenceId'])  
lable=tr\_mv\_df[['Sentiment']]  
tr\_mv\_df=tr\_mv\_df.drop(columns=['Sentiment'])  
te\_mv\_df = te\_mv\_df.drop(columns=['PhraseId', 'SentenceId'])  
  
tr\_mv\_df['Phrase'] = tr\_mv\_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))  
te\_mv\_df['Phrase'] = te\_mv\_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))  
  
max\_fets = 2000  
tokenizer = Tokenizer(num\_words=max\_fets, split=' ')  
tokenizer.fit\_on\_texts(tr\_mv\_df['Phrase'].values)  
X\_tr = tokenizer.texts\_to\_sequences(tr\_mv\_df['Phrase'].values)  
X\_tr = pad\_sequences(X\_tr)  
  
X\_tr.shape  
  
max\_fets = 1000  
tokenizer = Tokenizer(num\_words=max\_fets, split=' ')  
tokenizer.fit\_on\_texts(te\_mv\_df['Phrase'].values)  
X\_te = tokenizer.texts\_to\_sequences(te\_mv\_df['Phrase'].values)  
X\_te = pad\_sequences(X\_te)  
  
X\_te.shape  
  
embed\_dim = 128  
# lstm\_out = 196   
num\_classes = 5  
  
def create\_model\_plab2():  
 model\_plab2 = Sequential()  
 model\_plab2.add(Embedding(13734, embed\_dim, input\_length=X\_tr.shape[1]))  
 model\_plab2.add(Dropout(0.2))  
 model\_plab2.add(Conv1D(64, kernel\_size=3, padding='same', activation='relu', strides=1))  
 model\_plab2.add(GlobalMaxPooling1D())  
 model\_plab2.add(Dense(128, activation='relu'))  
 model\_plab2.add(Dropout(0.2))  
 model\_plab2.add(Dense(num\_classes, activation='softmax'))  
 model\_plab2.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  
 return model\_plab2  
  
lab\_enc = LableEncoder()  
integer\_encoded = lab\_enc.fit\_transform(lable)  
Y\_tr = to\_categorical(integer\_encoded)  
X\_tr, X\_te, Y\_tr, Y\_te = train\_test\_split(X\_tr, Y\_tr, test\_size=0.25, random\_state=30)  
print(X\_tr.shape,Y\_tr.shape)  
print(X\_te.shape,Y\_te.shape)  
  
model\_plab2 = create\_model\_plab2()  
  
history = model\_plab2.fit(X\_tr, Y\_tr, epochs=5, batch\_size=512, validation\_data=(X\_te,Y\_te))  
  
"""VALIDATION ON TEST DATA SET"""  
  
ypre=model\_plab2.predict\_classes(X\_te[:1])  
print(ypre[0]," PREDICTED LABLE")  
  
sub\_file = paid.read\_csv('sampleSubmission.csv',sep=',')  
print(sub\_file['Sentiment'].iloc[0]," ACTUAL LABLE")

Question 5:

import re  
import matplotlib.pyplot as plot  
import pandas as paid  
from keras import Sequential  
from keras.constraints import maxnorm  
from keras.layers import Embedding, Conv1D, Dropout, MaxPooling1D, Flatten, Dense, LSTM, SpatialDropout1D  
from keras.optimizers import SGD  
from keras.utils import to\_categorical  
from keras\_preprocessing.sequence import pad\_sequences  
from keras\_preprocessing.text import Tokenizer  
from sklearn.model\_plab2\_selection import train\_test\_split  
from sklearn.preprocessing import LableEncoder  
  
tr\_mv\_df = paid.read\_csv('train.tsv', delimiter='\t', encoding='utf-8')  
te\_mv\_df = paid.read\_csv('test.tsv', delimiter='\t', encoding='utf-8')  
  
tr\_mv\_df = tr\_mv\_df.drop(columns=['PhraseId', 'SentenceId'])  
lable=tr\_mv\_df[['Sentiment']]  
tr\_mv\_df=tr\_mv\_df.drop(columns=['Sentiment'])  
te\_mv\_df = te\_mv\_df.drop(columns=['PhraseId', 'SentenceId'])  
  
tr\_mv\_df['Phrase'] = tr\_mv\_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))  
te\_mv\_df['Phrase'] = te\_mv\_df['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))  
  
max\_fets = 2000  
tokenizer = Tokenizer(num\_words=max\_fets, split=' ')  
tokenizer.fit\_on\_texts(tr\_mv\_df['Phrase'].values)  
X\_tr = tokenizer.texts\_to\_sequences(tr\_mv\_df['Phrase'].values)  
X\_tr = pad\_sequences(X\_tr)  
  
X\_tr.shape  
  
max\_fets = 1000  
tokenizer = Tokenizer(num\_words=max\_fets, split=' ')  
tokenizer.fit\_on\_texts(te\_mv\_df['Phrase'].values)  
X\_te = tokenizer.texts\_to\_sequences(te\_mv\_df['Phrase'].values)  
X\_te = pad\_sequences(X\_te)  
  
X\_te.shape  
  
embed\_dim = 20  
lstm\_out = 10  
  
def create\_model\_plab2():  
 model\_plab2 = Sequential()  
 model\_plab2.add(Embedding(13734, embed\_dim, input\_length=X\_tr.shape[1]))  
 # model\_plab2.add(SpatialDropout1D(0.4))  
 model\_plab2.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2))  
 model\_plab2.add(Dense(5, activation='softmax'))  
 model\_plab2.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  
 return model\_plab2  
  
lab\_enc = LableEncoder()  
integer\_encoded = lab\_enc.fit\_transform(lable)  
Y\_tr = to\_categorical(integer\_encoded)  
X\_tr, X\_te, Y\_tr, Y\_te = train\_test\_split(X\_tr, Y\_tr, test\_size=0.25, random\_state=30)  
print(X\_tr.shape,Y\_tr.shape)  
print(X\_te.shape,Y\_te.shape)  
  
model\_plab2 = create\_model\_plab2()  
  
history = model\_plab2.fit(X\_tr, Y\_tr, epochs=5, batch\_size=40, validation\_data=(X\_te,Y\_te))  
  
"""PREDICTION ON TEST DATA SET"""  
  
ypre=model\_plab2.predict\_classes(X\_te[:1])  
print(ypre[0]," PREDICTED LABLE")  
  
sub\_file = paid.read\_csv('sampleSubmission.csv',sep=',')  
print(sub\_file['Sentiment'].iloc[0]," ACTUAL LABLE")

Question 6:

"""QUESTION 6: TUNE THE HYPER PARAMETERS"""  
  
from keras.optimizers import adam  
s=adam(lr=0.001)  
model\_plab21= Sequential()  
model\_plab21.add(Embedding(13734,50,input\_length=X\_tr.shape[1]))  
model\_plab21.add(Dropout(0.2))  
model\_plab21.add(Conv1D(64,kernel\_size=3,padding='same',activation='relu',strides=1))  
model\_plab21.add(GlobalMaxPooling1D())  
model\_plab21.add(Dense(128,activation='relu'))  
model\_plab21.add(Dropout(0.2))  
model\_plab21.add(Dense(num\_classes,activation='softmax'))  
model\_plab21.compile(loss='binary\_crossentropy',optimizer=s,metrics=['accuracy'])  
  
history1 = model\_plab21.fit(X\_tr, Y\_tr, epochs=5, batch\_size=512, validation\_data=(X\_te,Y\_te))

""QUESTION 6 : Changing HyperParameters"""  
  
from keras.optimizers import adam  
embed\_dim=50  
lstm\_out=50  
s=adam(lr=0.001)  
model\_plab21 = Sequential()  
model\_plab21.add(Embedding(13734, embed\_dim, input\_length = X\_tr.shape[1]))  
model\_plab21.add(LSTM(lstm\_out, dropout=0.2, recurrent\_dropout=0.2))  
model\_plab21.add(Dense(Y\_tr.shape[1],activation='softmax'))  
model\_plab21.compile(loss = 'binary\_crossentropy', optimizer=s,metrics = ['accuracy'])  
print(model\_plab21.summary())  
  
history1 = model\_plab21.fit(X\_tr, Y\_tr, epochs=5, batch\_size=50, validation\_data=(X\_te,Y\_te))

Question 7:

from keras.callbacks import TensorBoard  
from keras.layers import Input, Dense  
from keras.model\_plab2s import Model\_plab2  
  
# this is the size of our encoded representations  
encoding\_dim = 24  
# 32 floats -> compression of factor 24.5, assuming the input is 784 floats  
  
# this is our input placeholder  
input\_img = Input(shape=(784,))  
# "encoded" is the encoded representation of the input  
encoded = Dense(encoding\_dim, activation='relu')(input\_img)  
# "decoded" is the lossy reconstruction of the input  
decoded = Dense(784, activation='sigmoid')(encoded)  
# this model\_plab2 maps an input to its reconstruction  
autoencoder = Model\_plab2(input\_img, decoded)  
# seperate encoder model\_plab2  
# this model\_plab2 maps an input to its encoded representation  
encoder = Model\_plab2(input\_img, encoded)  
# create a seperate decoder model\_plab2  
# create a placeholder for an encoded (32-dimensional) input  
encoded\_input = Input(shape=(encoding\_dim,))  
# retrieve the last layer of the autoencoder model\_plab2  
decoder\_layer = autoencoder.layers[-1]  
# create the decoder model\_plab2  
decoder = Model\_plab2(encoded\_input, decoder\_layer(encoded\_input))  
autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy')  
  
from keras.datses import mnist  
import numpy asnupy  
  
(x\_tr,y\_tr), (x\_te,y\_te) = mnist.load\_data()  
x\_tr = x\_tr.astype('float32') / 255.  
x\_te = x\_te.astype('float32') / 255.  
x\_tr = x\_tr.reshape((len(x\_tr),nupy.prod(x\_tr.shape[1:])))  
x\_te = x\_te.reshape((len(x\_te),nupy.prod(x\_te.shape[1:])))  
# tensorboard = TensorBoard(write\_graph=True, log\_dir="/tmp/tensor-board/auto-encoders", histogram\_freq=0)  
autoencoder.fit(x\_tr, x\_tr,epochs=20,batch\_size=256,shuffle=True,verbose=1, validation\_data=(x\_te, x\_te))  
  
# encode and decode some digits  
# note that we take them from the \*test\* set  
encoded\_imgs = encoder.predict(x\_te)  
decoded\_imgs = decoder.predict(encoded\_imgs)  
  
# use Matplotlib  
import matplotlib.pyplot as pet  
  
# displaying original and reconstructed image  
n = 1 # how many images we will display  
pet.figure(figsize=(20, 4))  
for i in range(n):  
 # display original  
 ax = pet.subplot(2, n, i + 1)  
 pet.imshow(x\_te[i+4].reshape(28, 28))  
 pet.gray()  
 ax.get\_xaxis().set\_visible(False)  
 ax.get\_yaxis().set\_visible(False)  
  
 # display reconstruction  
 ax = pet.subplot(2, n, i + 1 + n)  
 pet.imshow(decoded\_imgs[i+4].reshape(28, 28))  
 pet.gray()  
 ax.get\_xaxis().set\_visible(False)  
 ax.get\_yaxis().set\_visible(False)  
pet.show()